LA-UR-02-4248

Title: PARSIM: Parallel Architecure Simulation Tool

Author(s): Nick Moss

Submitted to: LANL Student Symposium
Los Alamos, NM
August 2002

Los Alamos NATIONAL LABORATORY
Nick Moss
nickm@lanl.gov
CCS-3 / Modeling, Algorithms and Informatics Group

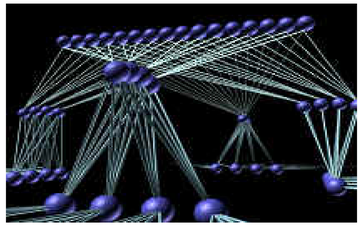
***PARSIM**: Parallel Architecture Simulation Tool*

Application and Scalability Issues in the Evaluation of ASCI/Extreme-scale Architectures

Abstract

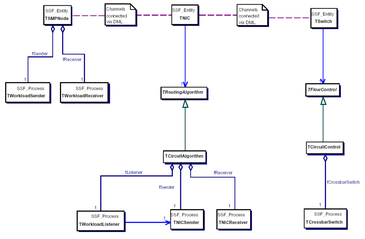
PARSIM, an extensible discrete-event simulation-based tool enables detailed analysis, evaluation, and modeling of existing and proposed massively-parallel computer architectures. Continuing refinements and development of higher-fidelity models expand predictive and modeling capability of ASCI and extreme-scale machines but present new challenges and greater system requirements. Scalability, execution requirements of large models with long time runs, verification, and model representation and generation are important factors which determine PARSIM's effectiveness.

DaSSF (Darthmouth Scalable Simulation Framework), provides an optimized discrete event framework for PARSIM. Modifications made to DaSSF expand platform-specific limitations and adapt DaSSF to the effective use of shared-memory in a distributed environment allowing the use of a 256 processor Alpha/Linux cluster. Memory requirements scale linearly with simulation time and portions of DaSSF's code reliant on 32-bit addressing have been modified to support 64-bit addressing to allow longer simulation times and a larger number of simulated nodes. An automated verification system using regression testing techniques helps ensure the system maintains consistency after each set of code changes and can be executed simultaneously using multiple test models and on a group of remote machines. DML (domain modeling language) allows for modularity in model generation and representation. Models may include various Quadrics hardware interconnects, switches, NIC's, SMP nodes arranged in fat-tree networks or unbalanced topologies that model their counterparts in current ASCI machines. A large set of components and their interconnection complexity stress the need for an alternative interface. A compact model description language provides a solution with improved handling in readability, creation, modication and structuring control.

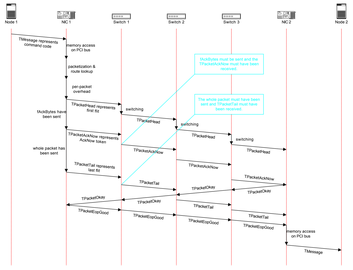


Framework and Implementation

Darthmouth Scalable Simulation Framework, a C++ implementation of SSF (Scalable Simulation Framework specification), provides an optimized discrete event and synchronization management system with support for shared-memory and distributed parallelism. PARSIM component classes define network components, routing methods, messages, packets and other constructs and inherit from the base classes defined by the SSF specification: Entity, Event, Process, and Input and Output Channels.



Class diagram of high-level PARSIM components



Message transmission through a PARSIM-simulated Quadrics network.

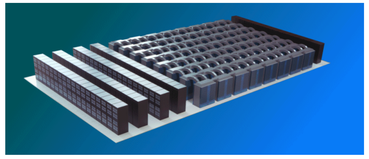
Scaling the Simulation Framework

-High fidelity models, particularly those using direct execution or emulated workloads and a high number of simulated nodes executed for long simulation times push the limits of available disk space, memory and computational time, requiring modifications to software, more hardware or both.

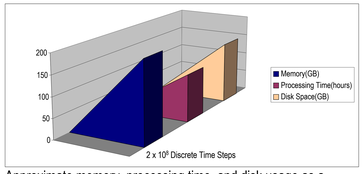
-A 64-node, 4 processor per node Alpha cluster utilizing Quadrics hardware and running dual boot Linux and TRU 64 is ideal for execution of large processor models but is not fully supported by the current DaSSF distribution.

-Portions of DaSSF related to shared-memory mapping and MPI interaction are modified to work with the cluster. Aside from the benefits of higher computational power, memory requirements are distributed, allowing a much larger memory space.

-DaSSF's reliance on 32-bit system calls in shared-memory mapping and other sections of code assumes 32-bit addressing, placing a limit of a 2 gigabyte heap per DaSSF process. Fortunately large models may be simulated in a highly distributed fashion at a low performance cost. For a 512 processor model, two simulated nodes may be assigned to each UNIX process. This configuration yields an effective address space of 512GB and allows simulations on the order of 1000+ processors with times exceeding 100,000,000 discrete time-steps.



High-fidelity extreme-scale(1000+ processor) simulations execute on a parallel machine of smaller size in order to meet memory requirements and reduce computational time.

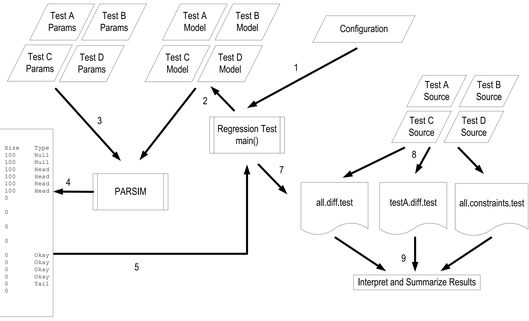


Approximate memory, processing time, and disk usage as a function of simulation time for a 512 processor statistical / uniform packet distribution model as executed on a 256 processor Alpha-Linux cluster.

Verification

-An automated regression testing system helps ensure the system maintains consistency after any changes are made to the code. The test program supports modular test models and scripts which can be executed in batch and on a group of remote machines.

-The testing mechanism uses predefined data files which are passed to an arbitrary number of test scripts which can then perform diff-matching, constraint verification, etc.. At the beginning of the process, PARSIM is built(2) with the current source files and executed with the test models(3) specified in the configuration(1). The output(5) is compared against the source data files for the specified model(8) and the script returns a boolean value and to indicate pass or fail along with additional test-specific information(9). Results are gathered and summarized for each test model and are used to interpret the net result.

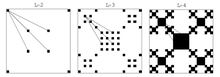


Execution output from test model X

```
ID     Time    Location        Action  Source  Target  Size    Type
0:1    10      0       MessageSent     0       1       100     Null
0:1    13      10000   MessageReceived 0       1       100     Null
0:1:1  526.513 10000   PacketSent      0       1       100     Head
0:1:1  543.513 20000   PacketReceived  0       1       100     Head
0:1:1  562.513 20000   PacketSent      0       Null    100     Head
0:1:1  575.513 10001   PacketReceived  0       Null    100     Head
0:1:1  686.114 10000   PacketSent      0       Null    0
AckNow
0:1:1  703.114 20000   PacketReceived  0       Null    0
AckNow
0:1:1  722.114 20000   PacketSent      0       Null    0
AckNow
0:1:1  735.114 10001   PacketReceived  0       Null    0
AckNow
0:1:1  735.114 10001   PacketSent      0       Null    0       Okay
0:1:1  752.114 20000   PacketReceived  0       Null    0       Okay
0:1:1  752.114 20000   PacketSent      0       Null    0       Okay
0:1:1  765.114 10000   PacketReceived  0       Null    0       Okay
0:1:1  785.864 10000   PacketSent      0       Null    0       Tail
0:1:1  785.864 10000   PacketSent      0       Null    0
EopGood
...
```

Machine Representation and DML Generation

-The interconnection networks from switch and node wiring results in lengthy DML code and poses difficulties in generation, modification, and readability. Other factors complicate management and require an automated creation approach for any model of significant size.



Fractal-tree representations of an L-level fat-tree network illustrate connection complexity as the number of layers and switches increases.

-A high-level model description abstracting wiring details in DML models provides a compact notation for specification and use of homogenuous and custom components.

-This approach allows for improved handling of large PARSIM models in readibility, creation, modification, and structure control.

-The description file is compiled into DML, preserving the ability to control portions of the model not parameterized in the description.

```
n.node.MESSAGE_DELAY     10
n.node.PACKET_DELAY      7
n.node.PACKET_SIZE       320
n.node.METHOD      Quadrics2
n.node.BUS_BANDWIDTH     200
n.nic.NET_BANDWIDTH      400
n.nic.ACK_BYTES          640
.
n.PACKET_DELAY     20
n.NET_BANDWIDTH    400
.
1:n[16]
2:n[16]
3:n[16]
4:n[64]
.
wire 11:12
wire 12:13 {1,0,2,3,4,5}
wire 13:14
```

Compile to DML

An idealized model description for a 3 switch-layer, 64 node machine. Here, the component properties are uniform. In real systems, interconnect properties may vary widely. The description language allows for precise control over each component while maintaining usability.

On compilation, the DML model is parsed and cast from C++ component classes.

PARSIM — Parse DML

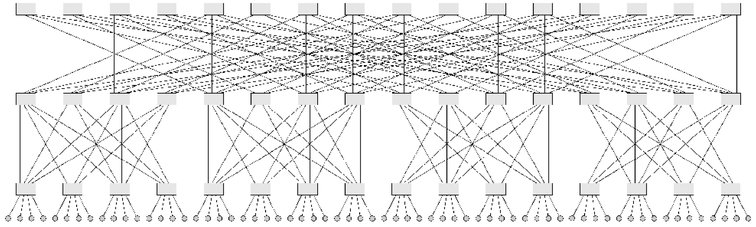
Build Executable

DML code parameterizes switches, nodes, and maps interconnections.

```
node [
  ID 0                  # use the convention that the relative entity idenity
                        # of an SMP node is always zero.
  INSTANCEOF "TSMPNode"
  PARAMS [
    STRING %i           # absolute entity identity
    INT    256          # mean size of messages (drawn at random from an
                        # exponential distribution)
    INT    100000       # mean delay between messages (drawn at random from an
                        # exponential distribution)
    INT    2000000      # the last time a message will be sent
  ]
]

# A network interface card.
nic [
  ID 1                  # use the convention that the relative entity
                        # identity of a network interface cars is always
# one
  INSTANCEOF "TNIC"
  PARAMS [
    STRING %i           # absolute entity identity
    STRING "CircuitAlgorithm" # the type of routing algorithm used
    INT    50           # delay between receipt of message and start of
                        # packet sending
    INT    1000         # delay between packets
    INT    320          # packet size
    INT    5000         # acknowledgement timeout
    STRING "Quadrics1" # routing method
  ]
]

# A switch.
switch [
  INSTANCEOF "TSwitch"
  PARAMS [
    STRING %i           # absolute entity identity
    STRING "CircuitControl" # the type of flow control used
    INT    8            # number of ports
    INT    35           # delay between receipt and retransmission of
                        # packets
  ]
...
MAP [FROM 5000.10000(LINKOUT4) TO 5000.20000(LINKIN0) DELAY 5]
MAP [FROM 5000.20000(LINKOUT0) TO 5000.10000(LINKIN4) DELAY 5]
MAP [FROM 5000.10000(LINKOUT5) TO 5000.20002(LINKIN0) DELAY 5]
MAP [FROM 5000.20001(LINKOUT0) TO 5000.10000(LINKIN5) DELAY 5]
MAP [FROM 5000.10002(LINKOUT6) TO 5000.20002(LINKIN0) DELAY 5]
MAP [FROM 5000.20002(LINKOUT0) TO 5000.10000(LINKIN6) DELAY 5]
MAP [FROM 5000.10000(LINKOUT7) TO 5000.20003(LINKIN0) DELAY 5]
MAP [FROM 5000.20003(LINKOUT0) TO 5000.10000(LINKIN7) DELAY 5]
MAP [FROM 5000.10001(LINKOUT4) TO 5000.20000(LINKIN1) DELAY 5]
MAP [FROM 5000.20000(LINKOUT1) TO 5000.10001(LINKIN4) DELAY 5]
MAP [FROM 5000.10001(LINKOUT5) TO 5000.20001(LINKIN1) DELAY 5]
MAP [FROM 5000.20001(LINKOUT1) TO 5000.10001(LINKIN5) DELAY 5]
MAP [FROM 5000.10002(LINKOUT6) TO 5000.20002(LINKIN1) DELAY 5]
MAP [FROM 5000.20002(LINKOUT1) TO 5000.10001(LINKIN6) DELAY 5]
MAP [FROM 5000.10001(LINKOUT7) TO 5000.20003(LINKIN1) DELAY 5]
MAP [FROM 5000.20003(LINKOUT1) TO 5000.10001(LINKIN7) DELAY 5]
...
```



Conclusion

The simulation of extreme-scale architectures presents significant demands in computational resources and challenges in complexity management. For models of increasing size and simulation time, high demands for memory, disk space, and computational time were solved by adapting the simulation framework to effective use of a shared-memory distributed environment. A specialized test system defines and implements a simplified interface that modularizes scripting, automates batch execution and data collection, speeding verification procedures that become time-consuming and difficult to manage with large models. For model representation, scaling presents a challenge in achieving both representational flexibility and usability. A condensed representation language provides an abstraction layer, compiling simplified models into native representation and finds a suitable balance of both.